

****

**特征点提取及相关系数法匹配**

**编程实习报告**

**实验时间： \*\*\*\* 年 \*\* 月 \*\* 日**

**学 院： 遥感信息工程学院**

**班 级： \*\*\*\***

**学 号： \*\*\*\*\*\*\*\*\*\*\*\*\***

**姓 名： \*\*\***

**实习地点： \*\*\*\*\*\***

***呵呵，你以为我会让你那么轻易知道我是谁吗***

## 概述

### 1、实习数据

实习用的是8位(灰度)航摄影像对，左影像是1023×942，右影像是805×887

### 2、编程环境

实习编程环境是VS2015+OpenCV 2.4.11+64位Debug平台

## 特征点提取算法原理及算法

原理：

角点被认为是在局部范围内与其周围点灰度值差异大的点。Moravec角点检测算子它是一种基于灰度方差的角点检测方法。该算子计算图像中某个像素点沿着水平、垂直、对角线、反对角线四个方向的灰度方差，其中的最小值选为该像素点的角点响应值IV(Interest Value)，因此IV越大与周围点灰度值差异就越大，最后再将一定大小的窗口内有最大的IV的点保留为角点。

算法：

void CMoravec::Moravec(Mat srcImage){//srcImage是输入的影像

//角点检测的结果图

resMorMat = srcImage.clone();

//存储兴趣点位置

Interest = Mat::ones(srcImage.rows, srcImage.cols, CV\_8U);

int r = kSize / 2;//kSize是兴趣值计算窗口

//获取图像的高和宽

const int nRows = srcImage.rows;

const int nCols = srcImage.cols;

int nCount = 0;

//保存角点的坐标

CvPoint \*pPoint = new CvPoint[nRows\*nCols];

//遍历图像

for (int i = r; i < srcImage.rows - r; i++)

{

for (int j = r; j < srcImage.cols - r; j++)

{

int wV1, wV2, wV3, wV4;

wV1 = wV2 = wV3 = wV4 = 0;

//计算水平方向窗内的兴趣值

for (int k = -r; k <= r; k++)

{

for (int m = -r; m <= r; m++)

{

//判断移动的过程中是否越界，越界的话就跳过当前的循环，以免出错

int a = i + k;

int b = j + m + 1;

if (b >= srcImage.cols)

{

continue;

}

wV1 += (srcImage.at<uchar>(i + k, j + m + 1) - srcImage.at<uchar>(i + k, j + m))

\*(srcImage.at<uchar>(i + k, j + m + 1) - srcImage.at<uchar>(i + k, j + m));

}

}

//计算垂直方向窗内的兴趣值

for (int k = -r; k <= r; k++)

{

for (int m = -r; m <= r; m++)

{

int a = i + k + 1;

int b = j + m;

if (a >= srcImage.rows)

{

continue;

}

wV2 += (srcImage.at<uchar>(i + k + 1, j + m) - srcImage.at<uchar>(i + k, j + m))

\*(srcImage.at<uchar>(i + k + 1, j + m) - srcImage.at<uchar>(i + k, j + m));

}

}

//计算45°方向窗内的兴趣值

for (int k = -r; k <= r; k++)

{

for (int m = -r; m <= r; m++)

{

int a = i + k + 1;

int b = j + m + 1;

if (a >= srcImage.rows || b >= srcImage.cols)

{

continue;

}

wV3 += (srcImage.at<uchar>(i + k + 1, j + m + 1) - srcImage.at<uchar>(i + k, j + m))

\*(srcImage.at<uchar>(i + k + 1, j + m + 1) - srcImage.at<uchar>(i + k, j + m));

}

}

//计算135°方向窗内的兴趣值

for (int k = -r; k <= r; k++)

{

for (int m = -r; m <= r; m++)

{

int a = i + k + 1;

int b = j + m - 1;

if (a >= srcImage.rows || b < 0)

{

continue;

}

wV4 += (srcImage.at<uchar>(a, b) - srcImage.at<uchar>(i + k, j + m))

\*(srcImage.at<uchar>(a, b) - srcImage.at<uchar>(i + k, j + m));

}

}

int value = min(min(wV1, wV2), min(wV3, wV4));

//如果兴趣值大于阈值，那么将坐标存入数组中

if (value > threshold)

{

pPoint[nCount] = cvPoint(j, i);

nCount++;

Interest.at<uchar>(i, j) = 0;

}

}

}

FeatureNum = nCount;

for (int i = 0; i < nCount; i++)

{

circle(resMorMat, pPoint[i], 5, Scalar(255, 0, 0),1,8,0);

}

}

## 相关系数法影像匹配原理及实用算法

影像匹配实质上是在两幅或是多幅影像之间识别同名点，其中基于相关系数的算法是实现影像匹配的基本算法之一。

相关系数是标准化的协方差函数，协方差函数除以两信号的方差即得相关系数。计算相关系数的公式如下：

g(x,y)与g’(x’,y’)的相关系数为：



其中：





若(p0, q0)>(p, q)，(p≠p0,q≠q0)，则p0,q0为搜索区影像相对于目标区影像的位移参数。由离散灰度数据对相关系数的估计为



，

考虑到计算工作量，相关系数的实用公式为



算法：

CMatch::CMatch()

{

windowsize = 11; //设置匹配窗口的大小

Threshold = 0.7; //设置阈值

Mpt2i = NULL;

num = 0;

}

算法1：相关系数计算

double CMatch::NCCScore(int lr, int lc, int rr, int rc)

{

double gLeftAverage = 0;//左影像窗口灰度平均值

double gRightAverage = 0;//右影像窗口灰度平均值

int halfsize = windowsize / 2;

for (int i = -halfsize; i<windowsize - halfsize; i++)

{

for (int j = -halfsize; j<windowsize - halfsize; j++)

{

gLeftAverage += LeftImgEpi.at<uchar>(lr + i, lc + j);

gRightAverage += RightImgEpi.at<uchar>(rr + i, rc + j);

}

}

gLeftAverage /= windowsize\*windowsize;

gRightAverage /= windowsize\*windowsize;

double a = 0;

double b = 0;

double c = 0;

for (int i = -halfsize; i<windowsize - halfsize; i++)

{

for (int j = -halfsize; j<windowsize - halfsize; j++)

{

double left\_av = LeftImgEpi.at<uchar>(lr + i, lc + j) - gLeftAverage;

double right\_av = RightImgEpi.at<uchar>(rr + i, rc + j) - gRightAverage;

a += left\_av\*right\_av;

b += left\_av\*left\_av;

c += right\_av\*right\_av;

}

}

return a / sqrt(b\*c);//返回相关系数的大小

}

算法2：图像拼接与匹配点搜索

void CMatch::FeatureMatchMain(Mat LeftImg, Mat RightImg, Mat LeftImgColor, Mat RightImgColor)

{

LeftImgEpi = LeftImg;

RightImgEpi = RightImg;

Point offset = Point(0, 170);

Mat Result;

showresult(LeftImgColor, RightImgColor, Result);//将两张彩色影像和合并1个

imshow("拼接结果", Result);

cvWaitKey();

Mat Interest;//兴趣矩阵

CMoravec CM;

int FeatureNum;//特征点个数

CM.Moravec(LeftImgEpi);//Moravec特征提取

Mat mFeaturePoint = CM.resMorMat.clone();

FeatureNum = CM.FeatureNum;

Interest = CM.Interest.clone();

Mpt2i = new struct MatchPt2i[FeatureNum];//给同名点结构体数组分配内存空间

int halfsize = windowsize / 2;

int Lr = LeftImgEpi.rows;

int Lc = LeftImgEpi.cols;

int Rr = RightImgEpi.rows;

int Rc = RightImgEpi.cols;

//搜索匹配点

for (int i = halfsize; i<Lr - halfsize; i++)

{

for (int j = 20; j<Lc - halfsize-1; j++)

{

if (Interest.at<uchar>(i, j) == 0)

//特征点作为模板中心

{

double maxscore = 0;

for (int r = i+ 5-20 ; r < i+5+ 20; r++){

for (int c = j +180 - 20 ; c< j+180 + 20; c++)

{

double score = NCCScore(i, j, r, c);//计算相关系数

if (score>maxscore)

{

maxscore = score;//计算相关系数的最大值

}

}

}

for (int r = i+5-20; r < i+5+20; r++){

for (int c = j+180- 20; c < Rr+20; c++)

{

double score = NCCScore(i, j, r, c);

if ((score == maxscore) && (score>Threshold))

{

//用直线连接同名点

line(Result, Point(j, i), Point(c + LeftImgEpi.cols, r), cvScalar(0, 0, 255));

//将匹配结果存入数组

Mpt2i[num].lpt = Point(j, i);

Mpt2i[num].rpt = Point(c, r);

num += 1;

}

}

}

}

}

}

imwrite("match\_result.bmp", Result);

imwrite("feature\_result.bmp", mFeaturePoint);

imshow("角点检测结果", mFeaturePoint);

imshow("匹配结果", Result);

//将匹配结果写入文件

FILE \*fp = fopen("match\_point.txt", "w");

if (fp == NULL)

{

return;

}

//匹配点对数

fprintf(fp, "%1i\n", num);

//存储格式：右影像列序号 右影像行序号 左影像列序号 左影像行序号

for (int i = 0; i<num; i++)

{

fprintf(fp, "%04d %04d %04d %04d\n",

Mpt2i[i].rpt.x, Mpt2i[i].rpt.y, Mpt2i[i].lpt.x, Mpt2i[i].lpt.y);

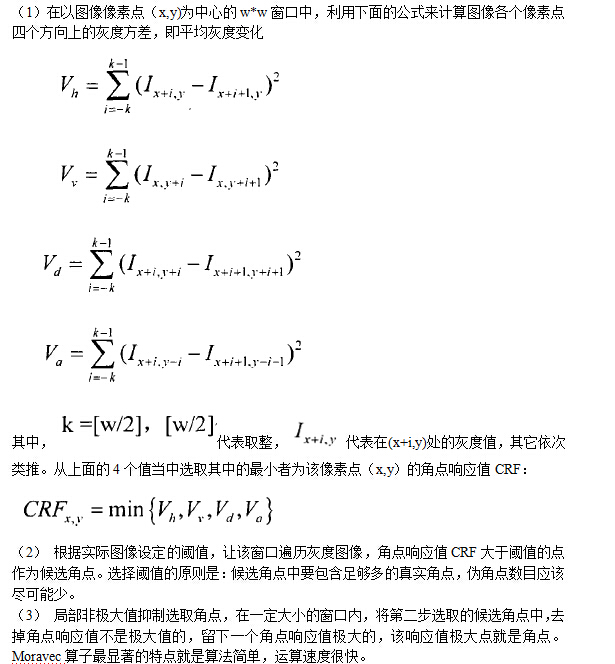
}

fclose(fp);

}

## 四、编程思路及流程图

### 1、Moravec点特征提取





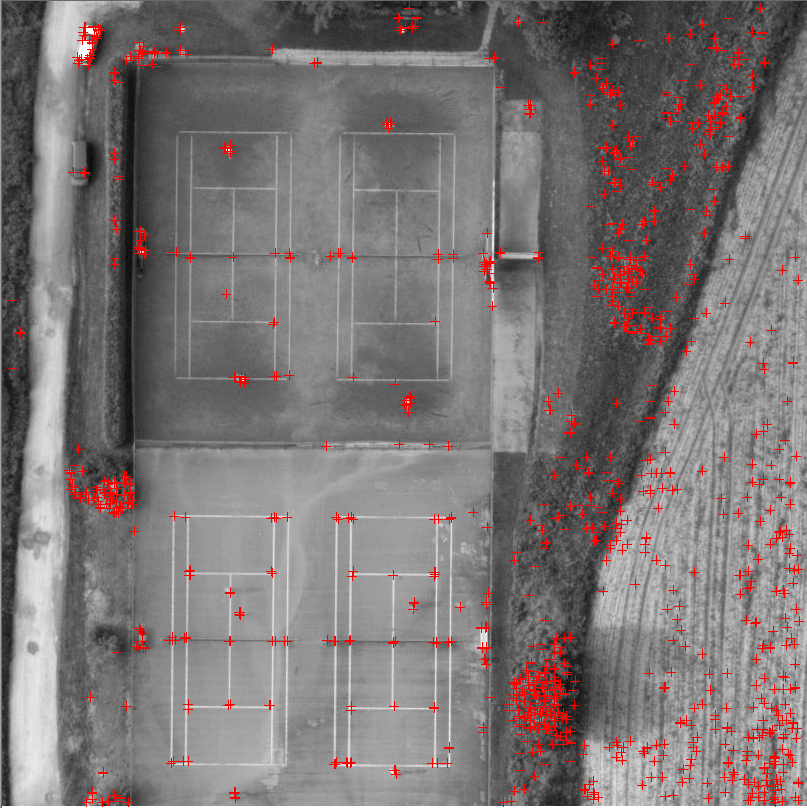
利用Moravec算子点特征提取流程图

### 2、相关系数法影像匹配

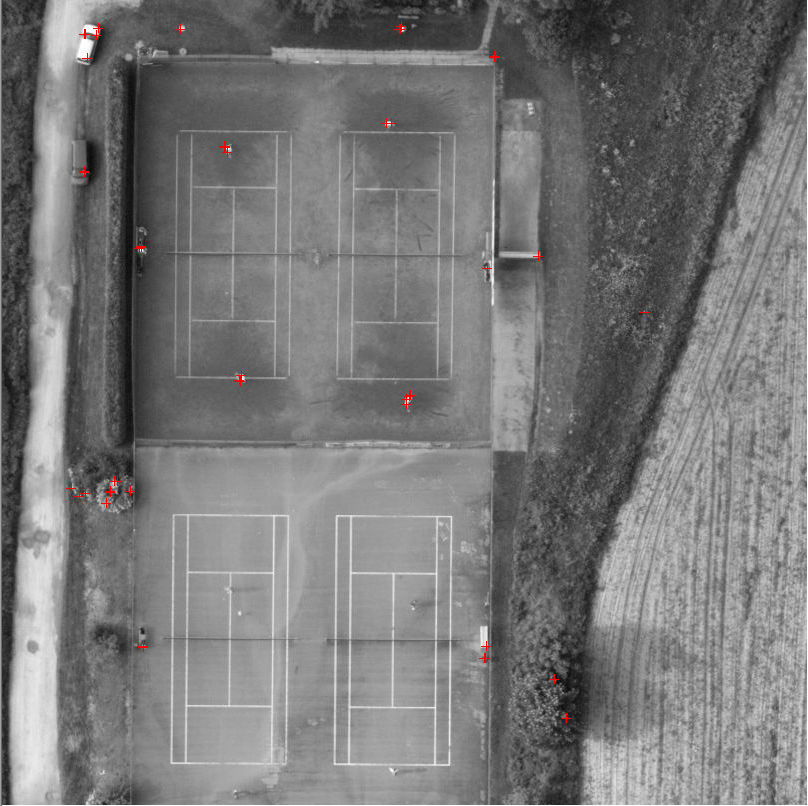


## 五、实验结果分析

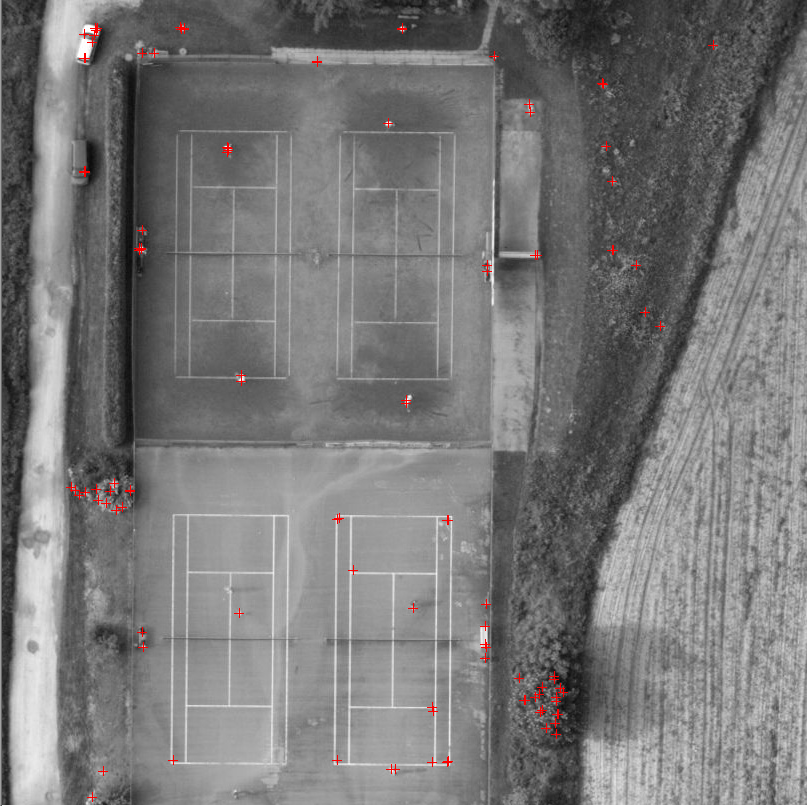
### 1、特征点提取算法阈值设定分析；随机分布与均匀分布提取特征点的实施分析



窗口5，阈值1000

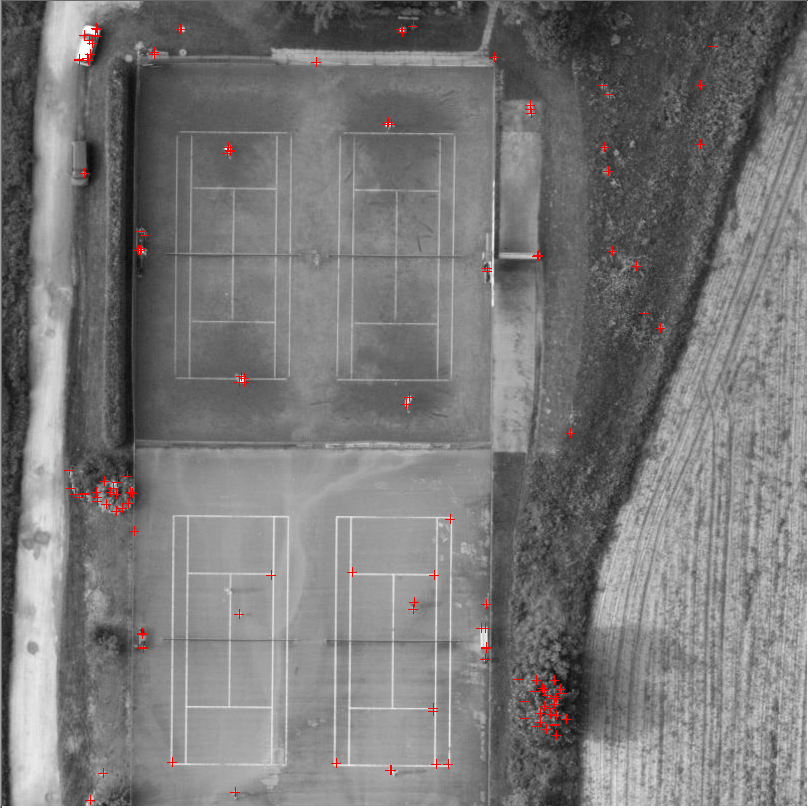


窗口5，阈值5000



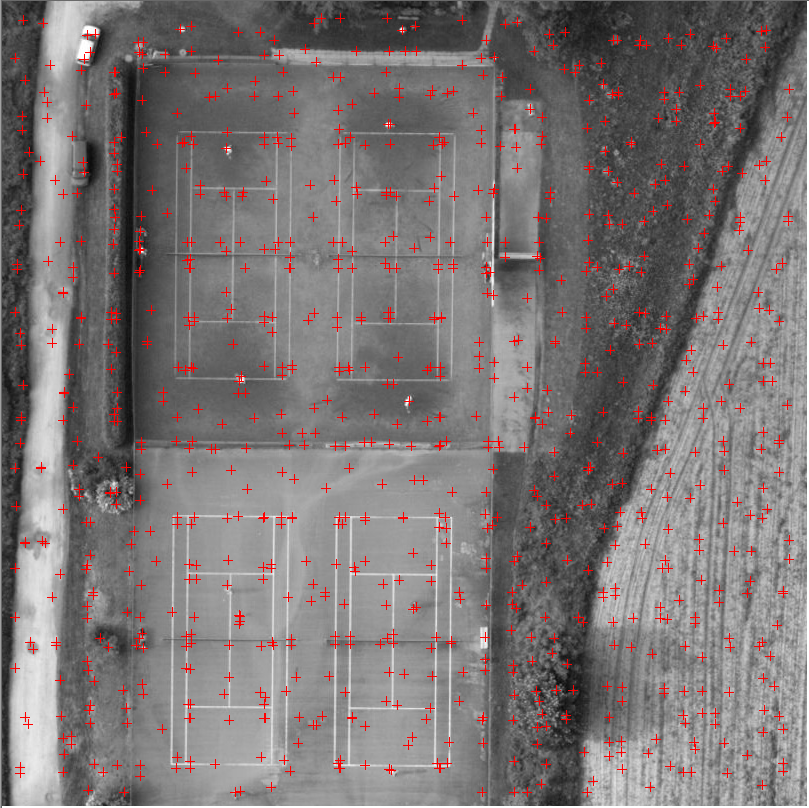
窗口11，阈值5000

可以看出阈值越大特征点越少，窗口越大特征点越多，合适的设置大约为窗口7，阈值3500

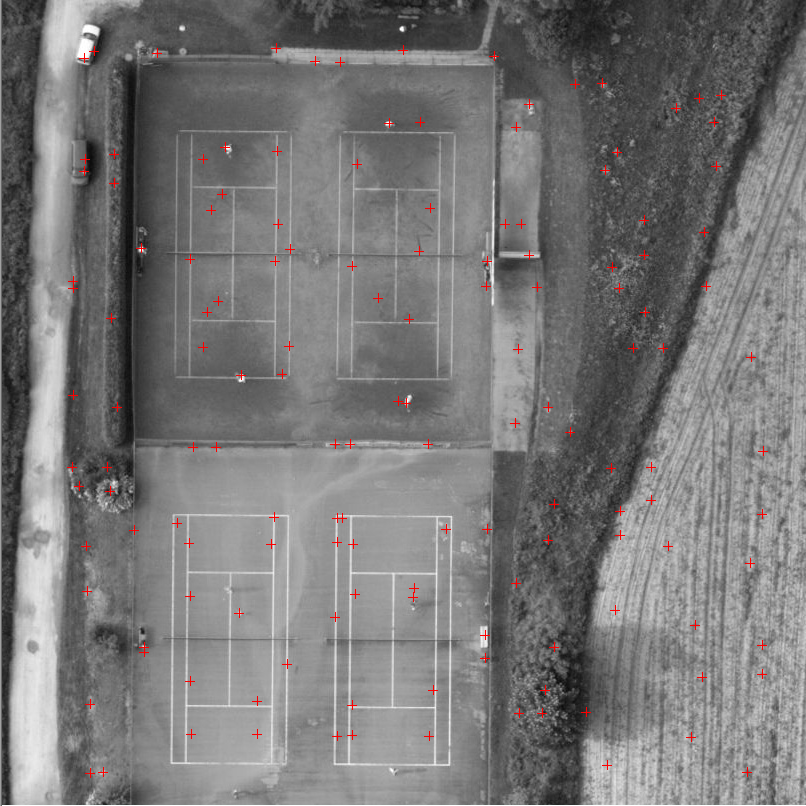


窗口7，阈值3500

上面采用的是随机分布的方法，如果采用均匀分布，效果就是下图这样。随机分布与均匀分布的控制完全是由阈值的设定做到的，随机分布提取出的兴趣值都要大于一个比较大的值，均匀分布的阈值是0，即一个格网中无论候选点兴趣值多大都会输出一个特征点



这是每25×25的窗口中都有一个特征点的效果



窗口：61×61

### 2、相关系数法影像匹配算法分析；窗口大小对匹配的影响

窗口大小会影响匹配的精度和速度。窗口越大，相关系数计算越复杂；搜索窗口越大，重复搜索的次数越多，两种都会导致匹配速度变慢。但是窗口越大，特别是搜索窗口越大，不一定导致匹配精度的提高，因为搜索窗口过大就可能包含相似的目标，容易导致匹配错误。这是个很难解决的问题，因为如果搜索窗口过小，由于经验视差并不精确，很模糊，搜索窗口过小就根本搜索不到正确匹配的目标，如果其中某个相关系数能越过阈值的限制，也会产生匹配错误，如下图方框中的错误就是窗口过小导致的。

